In [21]: **import** warnings **import** pandas **as** pd **import** seaborn **as** sns **import** matplotlib.pyplot **as** plt

**from** sklearn.preprocessing **import** StandardScaler **from** sklearn.ensemble **import** RandomForestRegressor **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.linear\_model **import** LinearRegression **from** sklearn.tree **import** DecisionTreeRegressor

**from** sklearn.ensemble **import** RandomForestRegressor, GradientBoostingRegre **from** sklearn.metrics **import** mean\_squared\_error, r2\_score **import** numpy **as** np

**from** sklearn.metrics **import** mean\_squared\_error, mean\_absolute\_error, r2\_s **from** sklearn.linear\_model **import** Lasso **from** xgboost **import** XGBRegressor

**from** sklearn.model\_selection **import** cross\_val\_score, KFold **from** sklearn.datasets **import** make\_regression

In [22]: pwd

Out[22]: 'C:\\Users\\Bennet\\AI PROJECT\_WORK'

In

[23]:

cd

AI

PROJECT\_WORK

[WinError 2] The system cannot find the file specified: 'AI PROJECT\_WOR K'

C:\Users\Bennet\AI PROJECT\_WORK

In

[24]:

pd

.

read\_csv

(

"AmesHousing.csv"

)

Out[24]: **MS MS Lot Lot Lot Land**

**Order PID Street Alley**

**SubClass Zoning Frontage Area Shape Contour**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 526301100 | 20 | RL | 141.0 | 31770 | Pave | NaN | IR1 | Lvl |
| **1** | 2 | 526350040 | 20 | RH | 80.0 | 11622 | Pave | NaN | Reg | Lvl |
| **2** | 3 | 526351010 | 20 | RL | 81.0 | 14267 | Pave | NaN | IR1 | Lvl |
| **3** | 4 | 526353030 | 20 | RL | 93.0 | 11160 | Pave | NaN | Reg | Lvl |
| **4** | 5 | 527105010 | 60 | RL | 74.0 | 13830 | Pave | NaN | IR1 | Lvl |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2925** | 2926 | 923275080 | 80 | RL | 37.0 | 7937 | Pave | NaN | IR1 | Lvl |
| **2926** | 2927 | 923276100 | 20 | RL | NaN | 8885 | Pave | NaN | IR1 | Low |
| **2927** | 2928 | 923400125 | 85 | RL | 62.0 | 10441 | Pave | NaN | Reg | Lvl |
| **2928** | 2929 | 924100070 | 20 | RL | 77.0 | 10010 | Pave | NaN | Reg | Lvl |
| **2929** | 2930 | 924151050 | 60 | RL | 74.0 | 9627 | Pave | NaN | Reg | Lvl |

2930 rows × 82 columns

In

[80]:

df

**=**

pd

.

read\_csv

(

"AmesHousing.csv"

)

<class 'pandas.core.frame.DataFrame'> RangeIndex: 2930 entries, 0 to 2929

In

[81]:

print

(

df

.

info

())

Data columns (total 82 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. Order 2930 non-null int64
2. PID 2930 non-null int64
3. MS SubClass 2930 non-null int64
4. MS Zoning 2930 non-null object
5. Lot Frontage 2440 non-null float64
6. Lot Area 2930 non-null int64
7. Street 2930 non-null object
8. Alley 198 non-null object
9. Lot Shape 2930 non-null object
10. Land Contour 2930 non-null object
11. Utilities 2930 non-null object
12. Lot Config 2930 non-null object
13. Land Slope 2930 non-null object
14. Neighborhood 2930 non-null object
15. Condition 1 2930 non-null object
16. Condition 2 2930 non-null object
17. Bldg Type 2930 non-null object
18. House Style 2930 non-null object
19. Overall Qual 2930 non-null int64
20. Overall Cond 2930 non-null int64
21. Year Built 2930 non-null int64
22. Year Remod/Add 2930 non-null int64
23. Roof Style 2930 non-null object
24. Roof Matl 2930 non-null object
25. Exterior 1st 2930 non-null object
26. Exterior 2nd 2930 non-null object
27. Mas Vnr Type 1155 non-null object
28. Mas Vnr Area 2907 non-null float64
29. Exter Qual 2930 non-null object
30. Exter Cond 2930 non-null object
31. Foundation 2930 non-null object
32. Bsmt Qual 2850 non-null object
33. Bsmt Cond 2850 non-null object
34. Bsmt Exposure 2847 non-null object
35. BsmtFin Type 1 2850 non-null object
36. BsmtFin SF 1 2929 non-null float64
37. BsmtFin Type 2 2849 non-null object
38. BsmtFin SF 2 2929 non-null float64
39. Bsmt Unf SF 2929 non-null float64
40. Total Bsmt SF 2929 non-null float64
41. Heating 2930 non-null object
42. Heating QC 2930 non-null object
43. Central Air 2930 non-null object
44. Electrical 2929 non-null object
45. 1st Flr SF 2930 non-null int64
46. 2nd Flr SF 2930 non-null int64
47. Low Qual Fin SF 2930 non-null int64
48. Gr Liv Area 2930 non-null int64
49. Bsmt Full Bath 2928 non-null float64
50. Bsmt Half Bath 2928 non-null float64
51. Full Bath 2930 non-null int64
52. Half Bath 2930 non-null int64
53. Bedroom AbvGr 2930 non-null int64
54. Kitchen AbvGr 2930 non-null int64
55. Kitchen Qual 2930 non-null object
56. TotRms AbvGrd 2930 non-null int64 56 Functional 2930 non-null object
    * 1. Fireplaces 2930 non-null int64
      2. Fireplace Qu 1508 non-null object
      3. Garage Type 2773 non-null object
      4. Garage Yr Blt 2771 non-null float64
      5. Garage Finish 2771 non-null object
      6. Garage Cars 2929 non-null float64
      7. Garage Area 2929 non-null float64
      8. Garage Qual 2771 non-null object
      9. Garage Cond 2771 non-null object
      10. Paved Drive 2930 non-null object
      11. Wood Deck SF 2930 non-null int64
      12. Open Porch SF 2930 non-null int64
      13. Enclosed Porch 2930 non-null int64
      14. 3Ssn Porch 2930 non-null int64
      15. Screen Porch 2930 non-null int64
      16. Pool Area 2930 non-null int64
      17. Pool QC 13 non-null object
      18. Fence 572 non-null object
      19. Misc Feature 106 non-null object
      20. Misc Val 2930 non-null int64
      21. Mo Sold 2930 non-null int64
      22. Yr Sold 2930 non-null int64
      23. Sale Type 2930 non-null object
      24. Sale Condition 2930 non-null object
      25. SalePrice 2930 non-null int64 dtypes: float64(11), int64(28), object(43)

memory usage: 1.8+ MB None

In [82]: print(df.describe())

0000

Overall Qual Overall Cond Year Built Year Remod/Add Mas V nr Area \

count 2930.000000 2930.000000 2930.000000 2930.000000 290

7.000000 mean 6.094881 5.563140 1971.356314 1984.266553 10

1.896801 std 1.411026 1.111537 30.245361 20.860286 17

9.112611 min 1.000000 1.000000 1872.000000 1950.000000

0.000000

25% 5.000000 5.000000 1954.000000 1965.000000

0.000000

50% 6.000000 5.000000 1973.000000 1993.000000

0.000000

75% 7.000000 6.000000 2001.000000 2004.000000 16

4.000000 max 10.000000 9.000000 2010.000000 2010.000000 160 0.000000

[83]: missing\_values **=** df.isnull().sum() print(missing\_values[missing\_values **>** 0])

Lot Frontage 490

Alley 2732

Mas Vnr Type 1775

Mas Vnr Area 23

Bsmt Qual 80

Bsmt Cond 80

Bsmt Exposure 83

BsmtFin Type 1 80

BsmtFin SF 1 1

BsmtFin Type 2 81

BsmtFin SF 2 1

Bsmt Unf SF 1

Total Bsmt SF 1

Electrical 1

Bsmt Full Bath 2

Bsmt Half Bath 2

Fireplace Qu 1422

Garage Type 157

Garage Yr Blt 159

Garage Finish 159

Garage Cars 1

Garage Area 1

Garage Qual 159

Garage Cond 159

Pool QC 2917

Fence 2358 Misc Feature 2824

dtype: int64

[84]:

print

(

df

.

columns

)

Index(['Order', 'PID', 'MS SubClass', 'MS Zoning', 'Lot Frontage', 'Lot Area',

'Street', 'Alley', 'Lot Shape', 'Land Contour', 'Utilities',

'Lot Config', 'Land Slope', 'Neighborhood', 'Condition 1',

'Condition 2', 'Bldg Type', 'House Style', 'Overall Qual',

'Overall Cond', 'Year Built', 'Year Remod/Add', 'Roof Style',

'Roof Matl', 'Exterior 1st', 'Exterior 2nd', 'Mas Vnr Type',

'Mas Vnr Area', 'Exter Qual', 'Exter Cond', 'Foundation', 'Bsmt Qual',

'Bsmt Cond', 'Bsmt Exposure', 'BsmtFin Type 1', 'BsmtFin SF 1',

'BsmtFin Type 2', 'BsmtFin SF 2', 'Bsmt Unf SF', 'Total Bsmt S F',

'Heating', 'Heating QC', 'Central Air', 'Electrical', '1st Flr S F',

'2nd Flr SF', 'Low Qual Fin SF', 'Gr Liv Area', 'Bsmt Full Bat h',

'Bsmt Half Bath', 'Full Bath', 'Half Bath', 'Bedroom AbvGr',

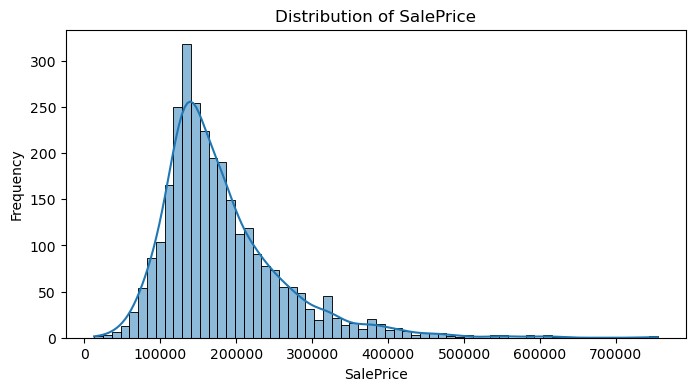
'Kitchen AbvGr', 'Kitchen Qual', 'TotRms AbvGrd', 'Functional',

'Fireplaces', 'Fireplace Qu', 'Garage Type', 'Garage Yr Blt',

'Garage Finish', 'Garage Cars', 'Garage Area', 'Garage Qual',

'Garage Cond', 'Paved Drive', 'Wood Deck SF', 'Open Porch SF'

[85]:



*# Suppress FutureWarnings*

warnings

.

simplefilter

(

action

**=**

'ignore'

,

category

**=**

FutureWarning

)

*# Example plot to check if the warning is suppressed*

plt

.

figure

(

figsize

**=**

(

8

,

4

))

sns

.

histplot

(

df\_reduced

[

'SalePrice'

]

,

kde

**=**

**True**

)

plt

.

title

(

'Distribution of SalePrice'

)

plt

.

xlabel

(

'SalePrice'

)

plt

.

ylabel

(

'Frequency'

)

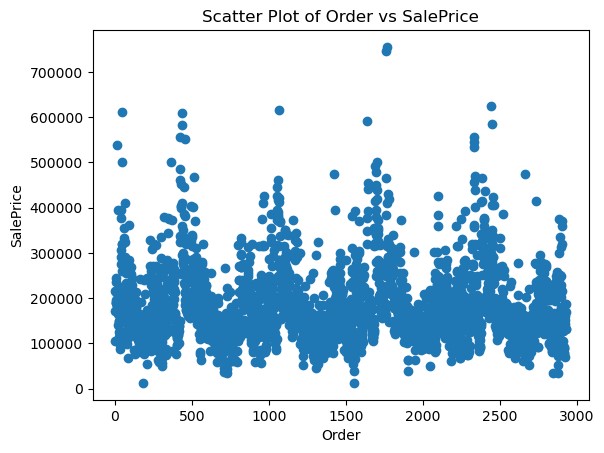
plt

.

show

()

[86]:



*# Print column names to verify*

plt

.

scatter

(

df

[

'Order'

]

,

df

[

'SalePrice'

])

*# Use the correct column name*

plt

.

xlabel

(

'Order'

)

plt

.

ylabel

(

'SalePrice'

)

plt

.

title

(

'Scatter Plot of Order vs SalePrice'

)

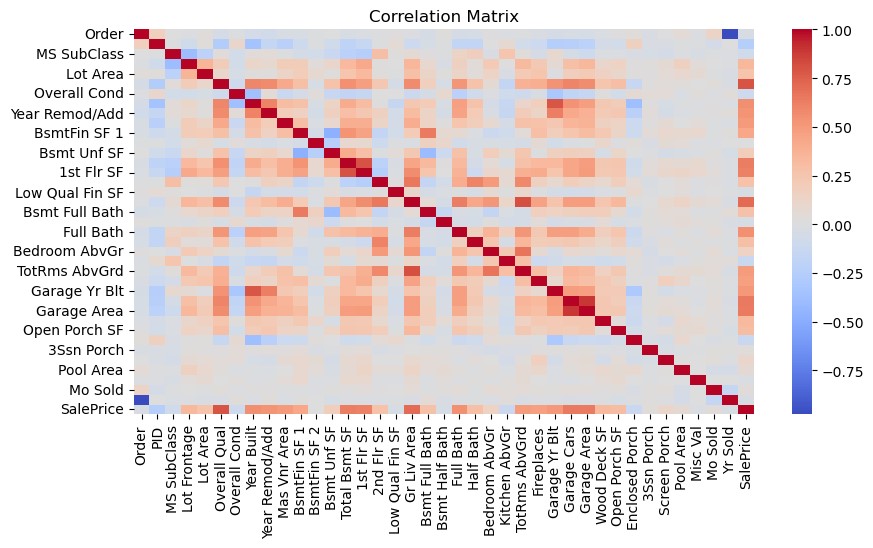
plt

.

show

()

[87]:



*# Select numeric columns only*

numeric\_df

**=**

df

.

select\_dtypes

(

include

**=**

[

'number'

])

*# Fill missing values with the median value of each column*

numeric\_df

**=**

numeric\_df

.

fillna

(

numeric\_df

.

median

())

*# Calculate the correlation matrix*

correlation\_matrix

**=**

numeric\_df

.

corr

()

*# Plot the correlation matrix*

plt

.

figure

(

figsize

**=**

(

10

,

5

))

sns

.

heatmap

(

correlation\_matrix

,

annot

**=**

**False**

,

cmap

**=**

'coolwarm'

)

plt

.

title

(

'Correlation Matrix'

)

plt

.

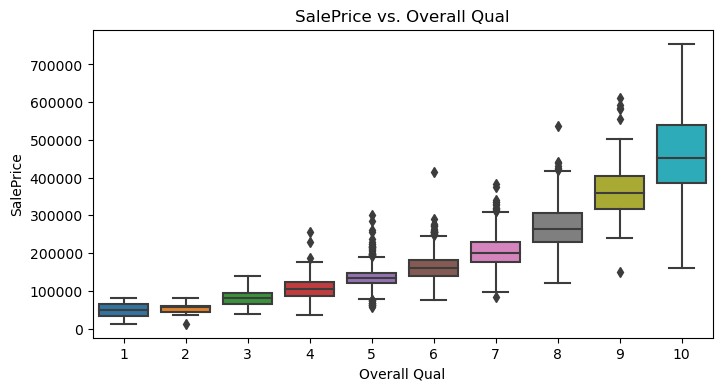
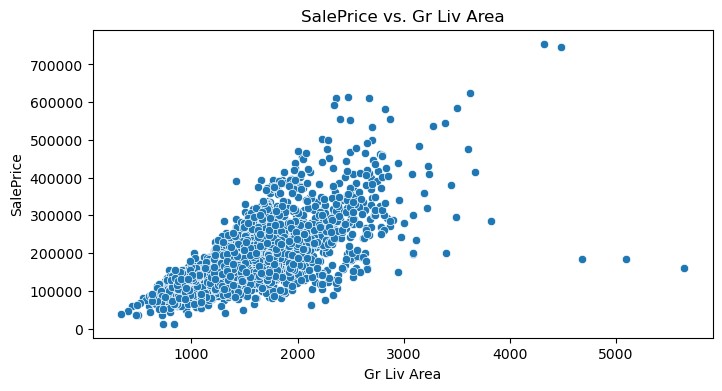
show

()

[42]:

In

[43]:



*# Suppress FutureWarnings*

warnings

.

simplefilter

(

action

**=**

'ignore'

,

category

**=**

FutureWarning

)

plt

.

figure

(

figsize

**=**

(

8

,

4

))

sns

.

scatterplot

(

x

**=**

'Gr Liv Area'

,

y

**=**

'SalePrice'

,

data

**=**

df

)

plt

.

title

(

'SalePrice vs. Gr Liv Area'

)

plt

.

show

()

*# Suppress FutureWarnings*

warnings

.

simplefilter

(

action

**=**

'ignore'

,

category

**=**

FutureWarning

)

plt

.

figure

(

figsize

**=**

(

8

,

4

))

sns

.

boxplot

(

x

**=**

'Overall Qual'

,

y

**=**

'SalePrice'

,

data

**=**

df

)

plt

.

title

(

'SalePrice vs. Overall Qual'

)

plt

.

show

()

[44]: *# Drop columns with more than 50% missing values* threshold **=** len(df) **\*** 0.5

df\_reduced **=** df.dropna(thresh**=**threshold, axis**=**1)

*# Check remaining columns with missing values* remaining\_missing **=** df\_reduced.isnull().sum()

remaining\_missing **=** remaining\_missing[remaining\_missing **>** 0].sort\_values remaining\_missing

Out[44]: Fireplace Qu 1422 Lot Frontage 490

Garage Qual 159

Garage Finish 159

Garage Yr Blt 159

Garage Cond 159

Garage Type 157

Bsmt Exposure 83

BsmtFin Type 2 81

BsmtFin Type 1 80

Bsmt Cond 80

Bsmt Qual 80

Mas Vnr Area 23

Bsmt Full Bath 2

Bsmt Half Bath 2

BsmtFin SF 1 1

BsmtFin SF 2 1

Bsmt Unf SF 1

Total Bsmt SF 1

Garage Cars 1

Garage Area 1 Electrical 1 dtype: int64

In [45]: *# Fill missing values for numerical columns with the median* **for** col **in** df\_reduced.select\_dtypes(include**=**['number']).columns: df\_reduced.loc[:, col] **=** df\_reduced[col].fillna(df\_reduced[col].media

*# Fill missing values for categorical columns with the mode* **for** col **in** df\_reduced.select\_dtypes(include**=**['object']).columns: df\_reduced.loc[:, col] **=** df\_reduced[col].fillna(df\_reduced[col].mode

*# Verify that there are no missing values left*

remaining\_missing\_after\_imputation **=** df\_reduced.isnull().sum() remaining\_missing\_after\_imputation **=** remaining\_missing\_after\_imputation[r print(remaining\_missing\_after\_imputation)

Series([], dtype: int64)

[46]: *# Identify categorical columns* categorical\_columns **=** df.select\_dtypes(include**=**['object']).columns categorical\_columns

Out[46]: Index(['MS Zoning', 'Street', 'Alley', 'Lot Shape', 'Land Contour',

'Utilities', 'Lot Config', 'Land Slope', 'Neighborhood', 'Condit ion 1',

'Condition 2', 'Bldg Type', 'House Style', 'Roof Style', 'Roof M atl',

'Exterior 1st', 'Exterior 2nd', 'Mas Vnr Type', 'Exter Qual',

'Exter Cond', 'Foundation', 'Bsmt Qual', 'Bsmt Cond', 'Bsmt Expo sure',

'BsmtFin Type 1', 'BsmtFin Type 2', 'Heating', 'Heating QC',

'Central Air', 'Electrical', 'Kitchen Qual', 'Functional',

'Fireplace Qu', 'Garage Type', 'Garage Finish', 'Garage Qual',

'Garage Cond', 'Paved Drive', 'Pool QC', 'Fence', 'Misc Featur e',

'Sale Type', 'Sale Condition'],

In

[47]:

dtype='object')

*# Apply one-hot encoding to the categorical columns*

df\_encoded

**=**

pd

.

get\_dummies

(

df

,

columns

**=**

categorical\_columns

,

drop\_first

**=**

**T**

*# Display the first few rows of the encoded dataframe*

df\_encoded

.

head

()

Out[47]: **Mas**

**MS Lot Lot Overall Overall Year Year**

**Order PID Vnr**

**SubClass Frontage Area Qual Cond Built Remod/Add**

**Area**

1. 1 526301100 20 141.0 31770 6 5 1960 1960 112.0
2. 2 526350040 20 80.0 11622 5 6 1961 1961 0.0
3. 3 526351010 20 81.0 14267 6 6 1958 1958 108.0
4. 4 526353030 20 93.0 11160 7 5 1968 1968 0.0
5. 5 527105010 60 74.0 13830 5 5 1997 1998 0.0
6. rows × 263 columns

In [48]: [49]: *# Identify numerical columns* numerical\_columns **=** df\_encoded.select\_dtypes(include**=**['float64', 'int64'] print("Numerical columns:\n", numerical\_columns)

encoded\_file\_path

**=**

'Encoded\_AmesHousing.csv'

*# Update this with the cor*

df\_encoded

.

to\_csv

(

encoded\_file\_path

,

index

**=**

**False**

)

*# Apply standardization to the numerical columns* scaler **=** StandardScaler()

df\_encoded[numerical\_columns] **=** scaler.fit\_transform(df\_encoded[numerical

*# Display the first few rows of the standardized dataframe* df\_encoded.head()

*# Save the standardized dataframe to a new CSV file*

standardized\_file\_path **=** 'Standardized\_AmesHousing.csv' *# Update this wi* df\_encoded.to\_csv(standardized\_file\_path, index**=False**)

Numerical columns:

Index(['Order', 'PID', 'MS SubClass', 'Lot Frontage', 'Lot Area',

'Overall Qual', 'Overall Cond', 'Year Built', 'Year Remod/Add', 'Mas Vnr Area', 'BsmtFin SF 1', 'BsmtFin SF 2', 'Bsmt Unf SF',

'Total Bsmt SF', '1st Flr SF', '2nd Flr SF', 'Low Qual Fin SF',

'Gr Liv Area', 'Bsmt Full Bath', 'Bsmt Half Bath', 'Full Bath',

'Half Bath', 'Bedroom AbvGr', 'Kitchen AbvGr', 'TotRms AbvGrd',

'Fireplaces', 'Garage Yr Blt', 'Garage Cars', 'Garage Area',

'Wood Deck SF', 'Open Porch SF', 'Enclosed Porch', '3Ssn Porch',

'Screen Porch', 'Pool Area', 'Misc Val', 'Mo Sold', 'Yr Sold',

'SalePrice'], dtype='object')

In [50]: *# Identify categorical columns* categorical\_columns **=** df.select\_dtypes(include**=**['object']).columns

*# Apply one-hot encoding to the categorical columns*

df\_encoded **=** pd.get\_dummies(df, columns**=**categorical\_columns, drop\_first**=T**

*# Identify numerical columns* numerical\_columns **=** df\_encoded.select\_dtypes(include**=**['float64', 'int64']

In [51]: *# Apply standardization to the numerical columns* scaler **=** StandardScaler()

df\_encoded[numerical\_columns] **=** scaler.fit\_transform(df\_encoded[numerical

*# Separate the target variable (SalePrice) from the features*

X **=** df\_encoded.drop(columns**=**['SalePrice']) y **=** df\_encoded['SalePrice']

In [52]: *# Drop columns with more than 50% missing values* threshold **=** len(df) **\*** 0.5

df\_reduced **=** df.dropna(thresh**=**threshold, axis**=**1)

[53]: remaining\_missing **=** df\_reduced.isnull().sum() remaining\_missing **=** remaining\_missing[remaining\_missing **>** 0].sort\_values print(remaining\_missing)

Fireplace Qu 1422

Lot Frontage 490

Garage Qual 159

Garage Finish 159

Garage Yr Blt 159

Garage Cond 159

Garage Type 157

Bsmt Exposure 83

BsmtFin Type 2 81

BsmtFin Type 1 80

Bsmt Cond 80

Bsmt Qual 80

Mas Vnr Area 23

Bsmt Full Bath 2

Bsmt Half Bath 2

BsmtFin SF 1 1

BsmtFin SF 2 1

Bsmt Unf SF 1

Total Bsmt SF 1

Garage Cars 1

Garage Area 1 Electrical 1

dtype: int64

Series([], dtype: int64)

[56]: *# Identify categorical columns* categorical\_columns **=** df\_reduced.select\_dtypes(include**=**['object']).column

*# Apply one-hot encoding to the categorical columns*

df\_encoded **=** pd.get\_dummies(df\_reduced, columns**=**categorical\_columns, drop

*# Identify numerical columns*

numerical\_columns **=** df\_encoded.select\_dtypes(include**=**['float64', 'int64']

*# Apply standardization to the numerical columns* scaler **=** StandardScaler()

df\_encoded[numerical\_columns] **=** scaler.fit\_transform(df\_encoded[numerical

*# Save the standardized dataframe to a new CSV file* standardized\_file\_path **=** 'Standardized\_AmesHousing.csv' df\_encoded.to\_csv(standardized\_file\_path, index**=False**)

*# Separate the target variable (SalePrice) from the features*

X **=** df\_encoded.drop(columns**=**['SalePrice']) y **=** df\_encoded['SalePrice']

In [57]: *# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2,

*# Train a Random Forest Regressor*

model **=** RandomForestRegressor(random\_state**=**42) model.fit(X\_train, y\_train)

*# Get feature importances*

feature\_importances **=** model.feature\_importances\_ features **=** X.columns

*# Create a DataFrame for feature importances*

importance\_df **=** pd.DataFrame({'Feature': features, 'Importance': feature\_ importance\_df **=** importance\_df.sort\_values(by**=**'Importance', ascending**=Fals**

*# Display the top 10 most important features* top\_features **=** importance\_df.head(10) print(top\_features)

Feature Importance

5 Overall Qual 0.604875

17 Gr Liv Area 0.102539

14 1st Flr SF 0.034611

13 Total Bsmt SF 0.027201

15 2nd Flr SF 0.023719

10 BsmtFin SF 1 0.022180

20 Full Bath 0.016725

4 Lot Area 0.016240

28 Garage Area 0.015769

27 Garage Cars 0.014788

[58]: *# Select the top 10 most important features* selected\_features **=** top\_features['Feature'].tolist()

*# Display the DataFrame with selected features* df\_selected **=** df\_encoded[selected\_features **+** ['SalePrice']] df\_selected.head()

*# Save the DataFrame with selected features to a new CSV file* selected\_features\_file\_path **=** 'Selected\_Features\_AmesHousing.csv' df\_selected.to\_csv(selected\_features\_file\_path, index**=False**)

In [59]: **!**pip install XGBoost

Requirement already satisfied: XGBoost in c:\users\bennet\anaconda3\new folder\new folder\lib\site-packages (2.1.0)

Requirement already satisfied: numpy in c:\users\bennet\anaconda3\new f older\new folder\lib\site-packages (from XGBoost) (1.24.3)

Requirement already satisfied: scipy in c:\users\bennet\anaconda3\new f older\new folder\lib\site-packages (from XGBoost) (1.11.3)

[60]: *# Suppress FutureWarnings* warnings.simplefilter(action**=**'ignore', category**=**FutureWarning)

*# Load the encoded and standardized data*

df\_encoded **=** pd.read\_csv('Standardized\_AmesHousing.csv')

*# Select the top 10 most important features plus SalePrice* selected\_features\_file\_path **=** 'Selected\_Features\_AmesHousing.csv' df\_selected **=** pd.read\_csv(selected\_features\_file\_path)

*# Separate the target variable (SalePrice) from the features*

X **=** df\_selected.drop(columns**=**['SalePrice']) y **=** df\_selected['SalePrice']

*# Split the data into training and testing sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2,

*# Function to evaluate the model* **def** evaluate\_model(model, X\_train, y\_train, X\_test, y\_test): model.fit(X\_train, y\_train) y\_pred\_train **=** model.predict(X\_train) y\_pred\_test **=** model.predict(X\_test)

print(f"{model.\_\_class\_\_.\_\_name\_\_}:")

print(f"Training RMSE: {mean\_squared\_error(y\_train, y\_pred\_train, squ print(f"Testing RMSE: {mean\_squared\_error(y\_test, y\_pred\_test, square print(f"Training R^2: {r2\_score(y\_train, y\_pred\_train)}") print(f"Testing R^2: {r2\_score(y\_test, y\_pred\_test)}") print("-" **\*** 50)

*# Linear Regression* lr\_model **=** LinearRegression()

evaluate\_model(lr\_model, X\_train, y\_train, X\_test, y\_test)

*# Decision Tree Regression*

dt\_model **=** DecisionTreeRegressor(random\_state**=**42) evaluate\_model(dt\_model, X\_train, y\_train, X\_test, y\_test)

*# Random Forest Regression*

rf\_model **=** RandomForestRegressor(random\_state**=**42) evaluate\_model(rf\_model, X\_train, y\_train, X\_test, y\_test)

*# Gradient Boosting Regression*

gb\_model **=** GradientBoostingRegressor(random\_state**=**42) evaluate\_model(gb\_model, X\_train, y\_train, X\_test, y\_test)

*# XGBoost Regression* **from** xgboost **import** XGBRegressor xgb\_model **=** XGBRegressor(random\_state**=**42)

evaluate\_model(xgb\_model, X\_train, y\_train, X\_test, y\_test)

LinearRegression:

Training RMSE: 0.4400271512444973

Testing RMSE: 0.48396196227145294

Training R^2: 0.7922439275128191

Testing R^2: 0.8136276316322577

--------------------------------------------------

DecisionTreeRegressor:

Training RMSE: 0.003100639816471489

Testing RMSE: 0.4671994415517178

Training R^2: 0.9999896843305761

Testing R^2: 0.8263144468107947

--------------------------------------------------

RandomForestRegressor:

Training RMSE: 0.13534481331808723

Testing RMSE: 0.3760618146352435

Training R^2: 0.9803447753603302

Testing R^2: 0.8874676104476606

--------------------------------------------------

GradientBoostingRegressor:

Training RMSE: 0.25572939234033204

Testing RMSE: 0.3846966429016676

Training R^2: 0.9298292577719908

Testing R^2: 0.8822405254768353

-------------------------------------------------XGBRegressor:

Training RMSE: 0.07121990301966404

Testing RMSE: 0.3820573362035425

Training R^2: 0.994557511339466

Testing R^2: 0.8838508186977561

--------------------------------------------------

In

[61]:

**!**

pip

install

scikit

**-**

learn

Requirement already satisfied: scikit-learn in c:\users\bennet\anaconda 3\new folder\new folder\lib\site-packages (1.3.0)

Requirement already satisfied: numpy>=1.17.3 in c:\users\bennet\anacond a3\new folder\new folder\lib\site-packages (from scikit-learn) (1.24.3) Requirement already satisfied: scipy>=1.5.0 in c:\users\bennet\anaconda 3\new folder\new folder\lib\site-packages (from scikit-learn) (1.11.3) Requirement already satisfied: joblib>=1.1.1 in c:\users\bennet\anacond a3\new folder\new folder\lib\site-packages (from scikit-learn) (1.2.0) Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\bennet

\anaconda3\new folder\new folder\lib\site-packages (from scikit-learn) (2.2.0)

In [62]: **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.datasets **import** make\_regression

*# Generate a random regression problem*

X, y **=** make\_regression(n\_samples**=**100, n\_features**=**20, noise**=**0.1, random\_st

*# Split the data into training and test sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2,

[63]: **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.datasets **import** make\_regression

*# Generate a random regression problem*

X, y **=** make\_regression(n\_samples**=**100, n\_features**=**20, noise**=**0.1, random\_st

*# Split the data into training and test sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2,

In [64]:

*# Example with XGBoost Regression* xgb\_model **=** XGBRegressor(random\_state**=**42) xgb\_metrics **=** evaluate\_model(xgb\_model, X\_train, y\_train, X\_test, y\_test)

XGBRegressor:

Training RMSE: 0.00047065222169994167

Testing RMSE: 104.61912194492157

Training R^2: 0.9999999999930302

Testing R^2: 0.33477901394185594

**def** evaluate\_model(model, X\_train, y\_train, X\_test, y\_test, cv\_folds**=**5):

*# Fit the model*

model.fit(X\_train, y\_train)

*# Make predictions*

y\_pred\_train **=** model.predict(X\_train) y\_pred\_test **=** model.predict(X\_test)

*# Calculate metrics for training set*

rmse\_train **=** np.sqrt(mean\_squared\_error(y\_train, y\_pred\_train)) mae\_train **=** mean\_absolute\_error(y\_train, y\_pred\_train) r2\_train **=** r2\_score(y\_train, y\_pred\_train)

*# Calculate metrics for test set*

rmse\_test **=** np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test)) mae\_test **=** mean\_absolute\_error(y\_test, y\_pred\_test) r2\_test **=** r2\_score(y\_test, y\_pred\_test)

print(f"Training set metrics:") print(f"RMSE: {rmse\_train}") print(f"MAE: {mae\_train}") print(f"R^2 Score: {r2\_train}")

print("\nTest set metrics:") print(f"RMSE: {rmse\_test}") print(f"MAE: {mae\_test}") print(f"R^2 Score: {r2\_test}")

*# Perform cross-validation*

kfold **=** KFold(n\_splits**=**cv\_folds, shuffle**=True**, random\_state**=**42) cv\_rmse\_scores **=** cross\_val\_score(model, X\_train, y\_train, cv**=**kfold, s cv\_mae\_scores **=** cross\_val\_score(model, X\_train, y\_train, cv**=**kfold, sc cv\_r2\_scores **=** cross\_val\_score(model, X\_train, y\_train, cv**=**kfold, sco

print("\nCross-validation metrics:")

print(f"RMSE: {**-**np.mean(cv\_rmse\_scores)} (+/- {np.std(cv\_rmse\_scores) print(f"MAE: {**-**np.mean(cv\_mae\_scores)} (+/- {np.std(cv\_mae\_scores)})" print(f"R^2 Score: {np.mean(cv\_r2\_scores)} (+/- {np.std(cv\_r2\_scores)

**return** {

"rmse\_train": rmse\_train,

"mae\_train": mae\_train, "r2\_train": r2\_train,

"rmse\_test": rmse\_test,

"mae\_test": mae\_test,

"r2\_test": r2\_test,

"cv\_rmse\_mean": **-**np.mean(cv\_rmse\_scores),

"cv\_rmse\_std": np.std(cv\_rmse\_scores),

"cv\_mae\_mean": **-**np.mean(cv\_mae\_scores),

"cv\_mae\_std": np.std(cv\_mae\_scores),

"cv\_r2\_mean": np.mean(cv\_r2\_scores),

"cv\_r2\_std": np.std(cv\_r2\_scores),

[65]:

}

[66]: *# Generate a random regression problem*

X, y **=** make\_regression(n\_samples**=**100, n\_features**=**20, noise**=**0.1, random\_st

*# Split the data into training and test sets*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2, \

[67]: *# Example with Lasso Regression* lasso\_model **=** Lasso(random\_state**=**42)

lasso\_metrics **=** evaluate\_model(lasso\_model, X\_train, y\_train, X\_test, y\_t

*# Example with XGBoost Regression* xgb\_model **=** XGBRegressor(random\_state**=**42)

xgb\_metrics **=** evaluate\_model(xgb\_model, X\_train, y\_train, X\_test, y\_test)

*# Print metrics for comparison* print("Lasso Regression Metrics:") print(lasso\_metrics)

print("\nXGBoost Regression Metrics:") print(xgb\_metrics)

Training set metrics:

RMSE: 3.317478165887205

MAE: 2.7578759207045356 R^2 Score: 0.9996537124931364

Test set metrics:

RMSE: 3.195632055564969

MAE: 2.688340930097737 R^2 Score: 0.9993793348822765

Cross-validation metrics:

RMSE: 4.010248653223467 (+/- 0.6547292667521755)

MAE: 3.3752269659799383 (+/- 0.539406342340989)

R^2 Score: 0.9994272599102466 (+/- 0.0002497415973117558)

Training set metrics:

RMSE: 0.00047065222169994167

MAE: 0.0003407340880874438 R^2 Score: 0.9999999999930302

Test set metrics:

RMSE: 104.61912194492157

MAE: 93.09415638646664 R^2 Score: 0.33477901394185594

Cross-validation metrics:

RMSE: 114.43743169776914 (+/- 24.664962356356806)

MAE: 92.40842299394453 (+/- 20.64241722478385) R^2 Score: 0.5286112439313368 (+/- 0.25735046937872297)

Lasso Regression Metrics:

{'rmse\_train': 3.317478165887205, 'mae\_train': 2.7578759207045356, 'r2\_ train': 0.9996537124931364, 'rmse\_test': 3.195632055564969, 'mae\_test': 2.688340930097737, 'r2\_test': 0.9993793348822765, 'cv\_rmse\_mean': 4.010 248653223467, 'cv\_rmse\_std': 0.6547292667521755, 'cv\_mae\_mean': 3.37522

69659799383, 'cv\_mae\_std': 0.539406342340989, 'cv\_r2\_mean': 0.999427259

9102466, 'cv\_r2\_std': 0.0002497415973117558}

XGBoost Regression Metrics:

{'rmse\_train': 0.00047065222169994167, 'mae\_train': 0.00034073408808744 38, 'r2\_train': 0.9999999999930302, 'rmse\_test': 104.61912194492157, 'm ae\_test': 93.09415638646664, 'r2\_test': 0.33477901394185594, 'cv\_rmse\_m ean': 114.43743169776914, 'cv\_rmse\_std': 24.664962356356806, 'cv\_mae\_me an': 92.40842299394453, 'cv\_mae\_std': 20.64241722478385, 'cv\_r2\_mean':

0.5286112439313368, 'cv\_r2\_std': 0.25735046937872297}

[68]:

Best Parameters: {'alpha': 0.01}

**from**

sklearn

.

model\_selection

**import**

GridSearchCV

**from**

sklearn

.

linear\_model

**import**

Lasso

*# Define the parameter grid*

param\_grid

**=**

{

'alpha'

[

:

0.01

,

0.1

,

1

,

10

,

100

]

*# Example values for Lasso's alpha*

}

*# Create a Lasso model*

lasso

**=**

Lasso

(

random\_state

**=**

42

)

*# Setup GridSearchCV*

grid\_search

**=**

GridSearchCV

(

estimator

**=**

lasso

,

param\_grid

**=**

param\_grid

,

scorin

*# Fit GridSearchCV*

grid\_search

.

fit

(

X\_train

,

y\_train

)

*# Best parameters and score*

print

(

f"Best Parameters:

{

grid\_search

.

best\_params\_

}

"

)

print

(

f"Best Score:

{

**-**

grid\_search

.

best\_score\_

}

"

)

Best Score: 0.10923982656926157

In [69]: **from** sklearn.model\_selection **import** GridSearchCV **from** xgboost **import** XGBRegressor *# Define the parameter grid* param\_grid **=** {

'n\_estimators': [100, 200, 300], *# Number of boosting rounds*

'learning\_rate': [0.01, 0.1, 0.2], *# Step size shrinkage*

'max\_depth': [3, 5, 7], *# Maximum depth of a tree*

'subsample': [0.8, 0.9, 1.0] *# Fraction of samples used for fitting* }

*# Create an XGBRegressor model* xgb **=** XGBRegressor(random\_state**=**42)

*# Setup GridSearchCV*

grid\_search **=** GridSearchCV(estimator**=**xgb, param\_grid**=**param\_grid, scoring**=**

*# Fit GridSearchCV*

grid\_search.fit(X\_train, y\_train)

*# Best parameters and score*

print(f"Best Parameters: {grid\_search.best\_params\_}") print(f"Best Score: {**-**grid\_search.best\_score\_}")

Best Parameters: {'learning\_rate': 0.1, 'max\_depth': 3, 'n\_estimators': 300, 'subsample': 0.8}

Best Score: 90.04136387833142

[71]: **from** sklearn.model\_selection **import** RandomizedSearchCV **from** sklearn.linear\_model **import** Lasso **from** scipy.stats **import** uniform *# Define the parameter distribution* param\_dist **=** {

'alpha': uniform(loc**=**0.01, scale**=**100) *# Randomly sample alpha values* }

*# Create a Lasso model* lasso **=** Lasso(random\_state**=**42)

*# Setup RandomizedSearchCV*

random\_search **=** RandomizedSearchCV(estimator**=**lasso, param\_distributions**=**p

*# Fit RandomizedSearchCV* random\_search.fit(X\_train, y\_train)

*# Best parameters and score*

print(f"Best Parameters: {random\_search.best\_params\_}") print(f"Best Score: {**-**random\_search.best\_score\_}")

Best Parameters: {'alpha': 5.818361216819946}

Best Score: 23.14106176849773

In [72]: **from** sklearn.model\_selection **import** RandomizedSearchCV **from** xgboost **import** XGBRegressor **from** scipy.stats **import** uniform, randint *# Define the parameter distribution* param\_dist **=** {

'n\_estimators': randint(100, 500), *# Randomly sample n\_estimators*

'learning\_rate': uniform(0.01, 0.2), *# Randomly sample learning\_rate*

'max\_depth': randint(3, 10), *# Randomly sample max\_depth*

'subsample': uniform(0.7, 0.3) *# Randomly sample subsample* }

In [73]: *# Create an XGBRegressor model* xgb **=** XGBRegressor(random\_state**=**42)

*# Setup RandomizedSearchCV*

random\_search **=** RandomizedSearchCV(estimator**=**xgb, param\_distributions**=**par

*# Fit RandomizedSearchCV* random\_search.fit(X\_train, y\_train)

*# Best parameters and score*

print(f"Best Parameters: {random\_search.best\_params\_}") print(f"Best Score: {**-**random\_search.best\_score\_}")

Best Parameters: {'learning\_rate': 0.13349630192554332, 'max\_depth': 4,

'n\_estimators': 121, 'subsample': 0.7021198915659151}

Best Score: 95.50956475483767

[74]:

Feature Importance

Feature\_7

0.279952

7

**import**

matplotlib

.

pyplot

**as**

plt

**import**

seaborn

**as**

sns

*# Extract feature importances*

feature\_importances

**=**

xgb\_model

.

feature\_importances\_

*# Define feature names manually if you don't have them*

feature\_names

**=**

[

f'Feature\_

{

i

}

'

**for**

i

**in**

range

(

X\_train

.

shape

[

1

])]

*# Create a DataFrame for plotting*

importances\_df

**=**

pd

.

DataFrame

({

'Feature'

:

feature\_names

,

'Importance'

:

feature\_importances

}).

sort\_values

(

by

**=**

'Importance'

,

ascending

**=**

**False**

)

*# Display top features*

print

(

importances\_df

.

head

(

10

))

8 Feature\_8 0.240319

1 Feature\_1 0.192401

9 Feature\_9 0.040291

4 Feature\_4 0.031751

11 Feature\_11 0.031567

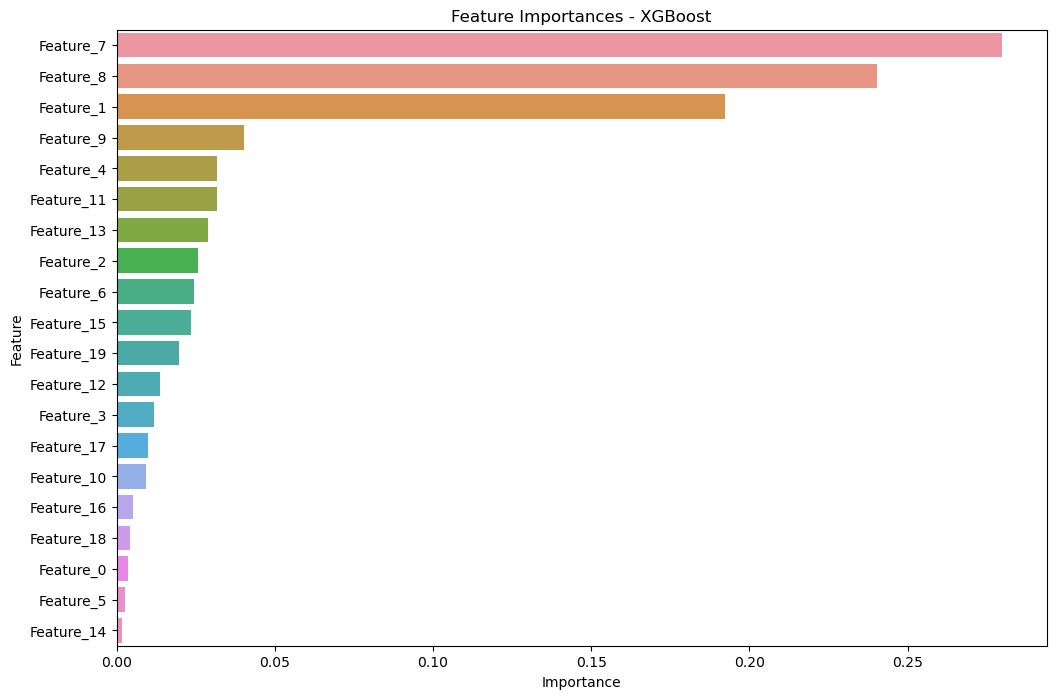
13 Feature\_13 0.028758

2 Feature\_2 0.025592

6 Feature\_6 0.024391

15 Feature\_15 0.023386

[75]:



*# Plot feature importances*

plt

.

figure

(

figsize

**=**

(

12

,

8

))

sns

.

barplot

(

x

**=**

'Importance'

,

y

**=**

'Feature'

,

data

**=**

importances\_df

)

plt

.

title

(

'Feature Importances - XGBoost'

)

plt

.

show

()

[76]:

*# Generate a sample dataset*

X, y **=** make\_regression(n\_samples**=**100, n\_features**=**20, noise**=**0.1, random\_st

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.2,

*# Train a Lasso model*

lasso\_model **=** Lasso(alpha**=**0.1, random\_state**=**42) lasso\_model.fit(X\_train, y\_train)

*# Extract coefficients*

lasso\_coefficients **=** lasso\_model.coef\_

*# Define feature names manually if you don't have them* feature\_names **=** [f'Feature\_{i}' **for** i **in** range(X\_train.shape[1])]

*# Create a DataFrame for plotting* lasso\_importances\_df **=** pd.DataFrame({ 'Feature': feature\_names,

'Coefficient': lasso\_coefficients

}).sort\_values(by**=**'Coefficient', ascending**=False**)

*# Display top features* print(lasso\_importances\_df.head(10))



Model: Linear Regression

Mean Absolute Error: 29685.15417133268

'R2'

:

r2

}

*# Display the results*

**for**

model\_name

,

metrics

**in**

results

.

items

():

print

(

f"Model:

{

model\_name

}

"

)

print

(

f"Mean Absolute Error:

{

metrics

[

'MAE'

]}

"

)

print

(

f"Mean Squared Error:

{

metrics

[

'MSE'

]}

"

)

print

(

f"R-squared:

{

metrics

[

'R2'

]}

"

)

print

()

Mean Squared Error: 2112266384.8901417

R-squared: 0.7365445890572946

Model: Random Forest

Mean Absolute Error: 15977.86196245734

Mean Squared Error: 707317345.1691883

R-squared: 0.9117788441971802

Model: Gradient Boosting

Mean Absolute Error: 15243.596848036725

Mean Squared Error: 666081972.994116 R-squared: 0.916921984285699